关于移动端适配基础概念，你必须要知道的！

移动端适配，是我们在开发中经常会遇到的，这里面可能会遇到非常多的问题：

- 1px 问题
- UI 图完美适配方案
- iPhoneX 适配方案
- 横屏适配
- 高清屏图片模糊问题
- ...

上面这些问题可能我们在开发中已经知道如何解决，但是问题产生的原理，以及解决方案的原理可能会模糊不清。在解决这些问题的过程中，我们往往会遇到非常多的概念：像素、分辨率、 PPI、 DPI、 DP、 DIP、 DPR、视口等等，你真的能分清这些概念的意义吗？

本文将从移动端适配的基础概念出发，探究移动端适配各种问题的解决方案和实现原理。

一、英寸

一般用英寸描述屏幕的物理大小，如电脑显示器的 17、 22，手机显示器的 4.8、 5.7 等使用的单位都是英寸。

需要注意，上面的尺寸都是屏幕对角线的长度：

英寸(inch,缩写为 in)在荷兰语中的本意是大拇指，一英寸就是指甲底部普通人拇指的宽度。

英寸和厘米的换算： 1 英寸=2.54 厘米

二、分辨率

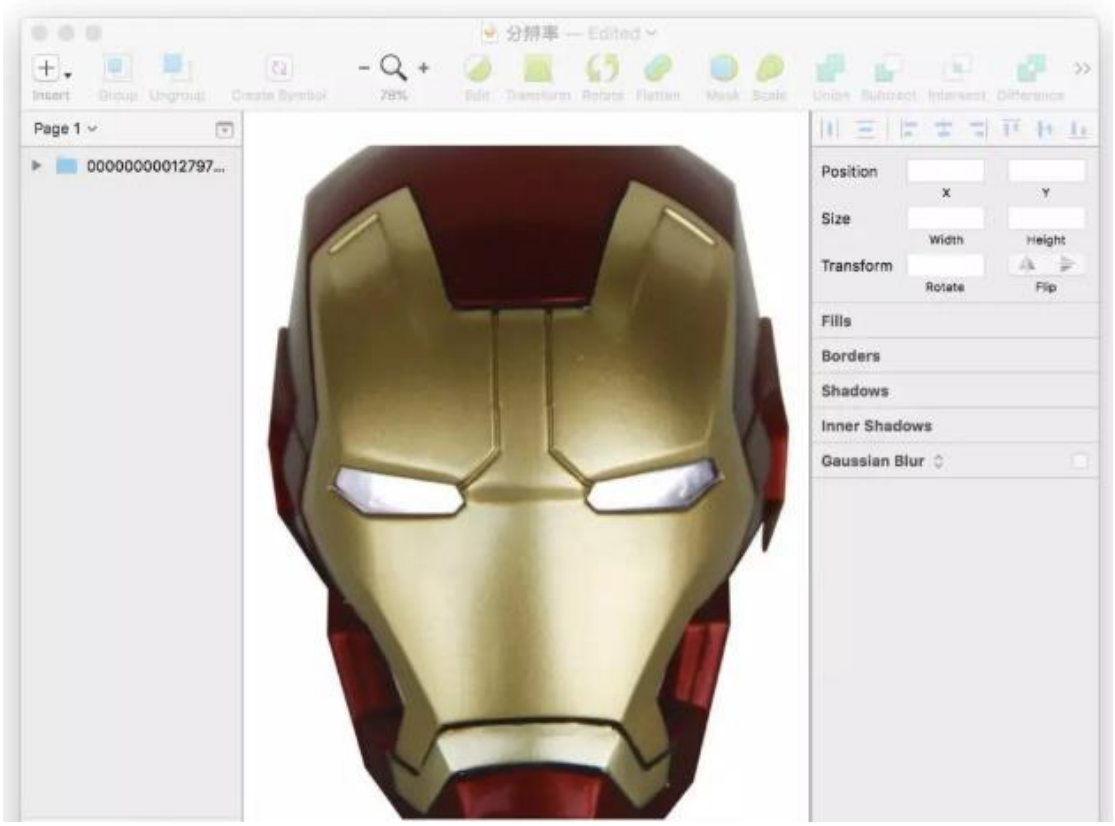
2.1 像素

像素即一个小方块，它具有特定的位置和颜色。

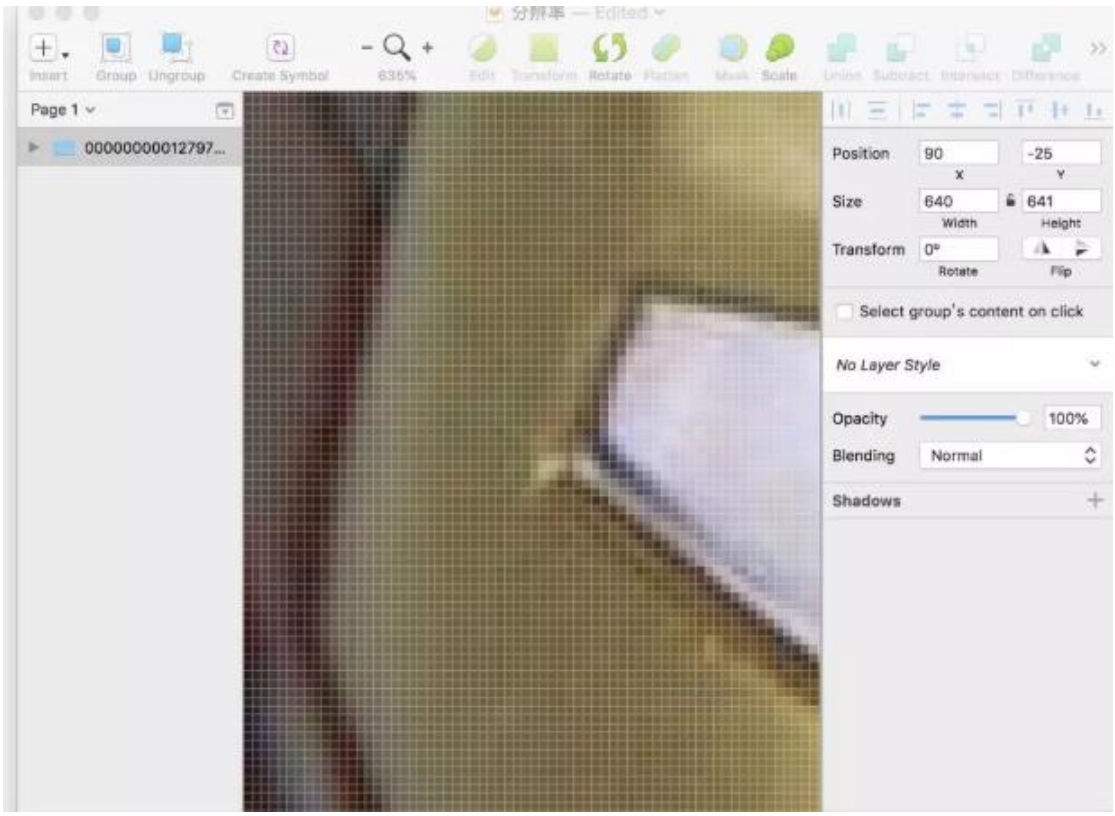
图片、电子屏幕（手机、电脑）就是由无数个具有特定颜色和特定位置的小方块拼接而成。

像素可以作为图片或电子屏幕的最小组成单位。

下面我们使用 `sketch` 打开一张图片：



将这些图片放大即可看到这些像素点：



通常我们所说的分辨率有两种，屏幕分辨率和图像分辨率。

2.2 屏幕分辨率

屏幕分辨率指一个屏幕具体由多少个像素点组成。

下面是 `apple` 的官网上对手机分辨率的描述：

`iPhone XSMax` 和 `iPhone SE` 的分辨率分别为 `2688x1242` 和 `1136x640`。这表示手机分别在垂直和水平上所具有的像素点数。

当然分辨率高不代表屏幕就清晰，屏幕的清晰程度还与尺寸有关。

2.3 图像分辨率

我们通常说的 图片分辨率其实是指图片含有的 像素数，比如一张图片的分辨率为 800x400。这表示图片分别在垂直和水平上所具有的像素点数为 800 和 400。

同一尺寸的图片，分辨率越高，图片越清晰。

2.4 PPI

PPI(PixelPerInch)：每英寸包括的像素数。

PPI 可以用于描述屏幕的清晰度以及一张图片的质量。

使用 PPI 描述图片时， PPI 越高，图片质量越高，使用 PPI 描述屏幕时，PPI 越高，屏幕越清晰。

在上面描述手机分辨率的图片中，我们可以看到： iPhone XSMax 和 iPhone SE 的 PPI 分别为 458 和 326，这足以证明前者的屏幕更清晰。

由于手机尺寸为手机对角线的长度，我们通常使用如下的方法计算 PPI:

$$ \frac{\sqrt{水平像素点数^2+垂直像素点数^2}}{尺寸}$$

iPhone6 的 PPI 为 $ \frac{\sqrt{1334^2+750^2}}{4.7}=325.6$，那它每英寸约含有 326 个物理像素点。

2.5 DPI

DPI(DotPerInch)：即每英寸包括的点数。

这里的点是一个抽象的单位，它可以是屏幕像素点、图片像素点也可以是打印机的墨点。

平时你可能会看到使用 DPI 来描述图片和屏幕，这时的 DPI 应该和 PPI 是等价的， DPI 最常用的是用于描述打印机，表示打印机每英寸可以打印的点数。

一张图片在屏幕上显示时，它的像素点数是规则排列的，每个像素点都有特定的位置和颜色。

当使用打印机进行打印时，打印机可能不会规则的将这些点打印出来，而是使用一个个打印点来呈现这张图像，这些打印点之间会有一定的空隙，这就是 DPI 所描述的：打印点的密度。

所以，打印机的 DPI 越高，打印图像的精细程度就越高，同时这也会消耗更多的墨点和时间。

三、设备独立像素

实际上，上面我们描述的像素都是 物理像素，即设备上真实的物理单元。

下面我们来看看 设备独立像素究竟是如何产生的：

智能手机发展非常之快，在几年之前，我们还用着分辨率非常低的手机，比如下面左侧的白色手机，它的分辨率是 320x480，我们可以在上面浏览正常的文字、图片等等。

但是，随着科技的发展，低分辨率的手机已经不能满足我们的需求了。很快，更高分辨率的屏幕诞生了，比如下面的黑色手机，它的分辨率是 640x940，正好是白色手机的两倍。

理论上来讲，在白色手机上相同大小的图片和文字，在黑色手机上会被缩放一倍，因为它的分辨率提高了一倍。这样，岂不是后面出现更高分辨率的手机，页面元素会变得越来越小吗？

然而，事实并不是这样的，我们现在使用的智能手机，不管分辨率多高，他们所展示的界面比例都是基本类似的。乔布斯在 iPhone4 的发布会上首次提出了 RetinaDisplay(视网膜屏幕)的概念，它正是解决了上面的问题，这也使它成为一款跨时代的手机。

在 iPhone4 使用的视网膜屏幕中，把 2x2 个像素当 1 个像素使用，这样让屏幕看起来更精致，但是元素的大小却不会改变。

如果黑色手机使用了视网膜屏幕的技术，那么显示结果应该是下面的情况，比如列表的宽度为 300 个像素，那么在一条水平线上，白色手机会用 300 个物理像素去渲染它，而黑色手机实际上会用 600 个物理像素去渲染它。

我们必须用一种单位来同时告诉不同分辨率的手机，它们在界面上显示元素的大小是多少，这个单位就是设备独立像素(DeviceIndependentPixels)简称 DIP 或 DP。上面我们说，列表的宽度为 300 个像素，实际上我们可以说：列表的宽度为 300 个设备独立像素。

打开 chrome 的开发者工具，我们可以模拟各个手机型号的显示情况，每种型号上面会显示一个尺寸，比如 iPhone X 显示的尺寸是 375x812，实际 iPhone X 的分辨率会比这高很多，这里显示的就是设备独立像素。

、

3.1 设备像素比

设备像素比 device pixel ratio 简称 dpr，即物理像素和设备独立像素的比值。

在 web 中，浏览器为我们提供了 window.devicePixelRatio 来帮助我们获取 dpr。

在 css 中，可以使用媒体查询 min-device-pixel-ratio，区分 dpr：

```
@media (-webkit-min-device-pixel-ratio: 2),(min-device-pixel-ratio: 2){ }
```

在 ReactNative 中，我们也可以使用 PixelRatio.get()来获取 DPR。

当然，上面的规则也有例外， iPhone6、7、8Plus 的实际物理像素是 1080x1920，在开发者工具中我们可以看到：它的设备独立像素是 414x736，设备像素比为 3，设备独立像素和设备像素比的乘积并不等于 1080x1920，而是等于 1242x2208。

实际上，手机会自动把 1242x2208 个像素点塞进 1080*1920 个物理像素点来渲染，我们不用关心这个过程，而 1242x2208 被称为屏幕的 设计像素。我们开发过程中也是以这个 设计像素为准。

实际上，从苹果提出视网膜屏幕开始，才出现设备像素比这个概念，因为在这之前，移动设备都是直接使用物理像素来进行展示。

紧接着， Android 同样使用了其他的技术方案来实现 DPR 大于 1 的屏幕，不过原理是类似的。由于 Android 屏幕尺寸非常多、分辨率高低跨度非常大，不像苹果只有它自己的几款固定设备、尺寸。所以，为了保证各种设备的显示效果， Android 按照设备的像素密度将设备分成了几个区间：

当然，所有的 Android 设备不一定严格按照上面的分辨率，每个类型可能对应几种不同分辨率，所以，每个 Android 手机都能根据给定的区间范围，确定自己的 DPR，从而拥有类似的显示。当然，仅仅是类似，由于各个设备的尺寸、分辨率上的差异，设备独立像素也不会完全相等，所以各种 Android 设备仍然不能做到在展示上完全相等。

3.2 移动端开发

在 iOS、 Android 和 ReactNative 开发中样式单位其实都使用的是设备独立像素。

iOS 的尺寸单位为 pt， Android 的尺寸单位为 dp， ReactNative 中没有指定明确的单位，它们其实都是设备独立像素 dp。

在使用 ReactNative 开发 App 时， UI 给我们的原型图一般是基于 iphone6 的像素给定的。

为了适配所有机型，我们在写样式时需要把物理像素转换为设备独立像素：例如：如果给定一个元素的高度为 200px(这里的 px 指物理像素，非 CSS 像素)， iphone6 的设备像素比为 2，我们给定的 height 应为 200px/2=100dp。

当然，最好的是，你可以和设计沟通好，所有的 UI 图都按照设备独立像素来出。

我们还可以在代码(ReactNative)中进行 px 和 dp 的转换：

```
1.  import {PixelRatio } from 'react-native';
2.
```

```
const dpr = PixelRatio.get();

/**
 * px 转换为 dp
 */
export function pxConvertTodp(px) {
  return px / dpr;
}

/**
 * dp 转换为 px
 */
export function dpConvertTopx(dp) {
  return PixelRatio.getPixelSizeForLayoutSize(dp);
}
```

3.3 WEB 端开发

在写 CSS 时，我们用到最多的单位是 px，即 CSS 像素，当页面缩放比例为 100%时，一个 CSS 像素等于一个设备独立像素。

但是 CSS 像素是很容易被改变的，当用户对浏览器进行了放大， CSS 像素会被放大，这时一个 CSS 像素会跨越更多的物理像素。

页面的缩放系数=CSS 像素/设备独立像素。

3.4 关于屏幕

这里多说两句 Retina 屏幕，因为我在很多文章中看到对 Retina 屏幕的误解。

Retina 屏幕只是苹果提出的一个营销术语：

> 在普通的使用距离下，人的肉眼无法分辨单个的像素点。

为什么强调 普通的使用距离下呢？我们来看一下它的计算公式：

$$ a=2arctan(h/2d) $$

a 代表人眼视角， h 代表像素间距， d 代表肉眼与屏幕的距离，符合以上条件的屏幕可以使肉眼看不见单个物理像素点。

它不能单纯的表达分辨率和 PPI，只能一种表达视觉效果。

让多个物理像素渲染一个独立像素只是 Retina 屏幕为了达到效果而使用的一种技术。而不是所有 DPR>1 的屏幕就是 Retina 屏幕。

比如：给你一块超大尺寸的屏幕，即使它的 PPI 很高， DPR 也很高，在近距离你也能看清它的像素点，这就不算 Retina 屏幕。

我们经常见到用 K 和 P 这个单位来形容屏幕：

P 代表的就是屏幕纵向的像素个数， 1080P 即纵向有 1080 个像素，分辨率为 1920X1080 的屏幕就属于 1080P 屏幕。

我们平时所说的高清屏其实就是屏幕的物理分辨率达到或超过 1920X1080 的屏幕。

K 代表屏幕横向有几个 1024 个像素，一般来讲横向像素超过 2048 就属于 2K 屏，横向像素超过 4096 就属于 4K 屏。

四、视口

视口(viewport)代表当前可见的计算机图形区域。在 Web 浏览器术语中，通常与浏览器窗口相同，但不包括浏览器的 UI， 菜单栏等——即指你正在浏览的文档的那一部分。

一般我们所说的视口共包括三种：布局视口、视觉视口和理想视口，它们在屏幕适配中起着非常重要的作用。

4.1 布局视口

布局视口(`layout viewport`)：当我们以百分比来指定一个元素的大小时，它的计算值是由这个元素的包含块计算而来的。当这个元素是最顶级的元素时，它就是基于布局视口来计算的。

所以，布局视口是网页布局的基准窗口，在`PC`浏览器上，布局视口就等于当前浏览器的窗口大小（不包括`borders`、`margins`、滚动条）。

在移动端，布局视口被赋予一个默认值，大部分为`980px`，这保证`PC`的网页可以在手机浏览器上呈现，但是非常小，用户可以手动对网页进行放大。

我们可以通过调用`document.documentElement.clientWidth/clientHeight`来获取布局视口大小。

4.2 视觉视口

视觉视口(`visual viewport`)：用户通过屏幕真实看到的区域。

视觉视口默认等于当前浏览器的窗口大小（包括滚动条宽度）。

当用户对浏览器进行缩放时，不会改变布局视口的大小，所以页面布局是不变的，但是缩放会改变视觉视口的大小。

例如：用户将浏览器窗口放大了 200%，这时浏览器窗口中的 CSS 像素会随着视觉视口的放大而放大，这时一个 CSS 像素会跨越更多的物理像素。

所以，布局视口会限制你的 CSS 布局而视觉视口决定用户具体能看到什么。

我们可以通过调用 window.innerWidth/innerHeight 来获取视觉视口大小。

4.3 理想视口

布局视口在移动端展示的效果并不是一个理想的效果，所以理想视口(ideal viewport)就诞生了：网站页面在移动端展示的理想大小。

如上图，我们在描述设备独立像素时曾使用过这张图，在浏览器调试移动端时页面上给定的像素大小就是理想视口大小，它的单位正是设备独立像素。

上面在介绍 CSS 像素时曾经提到 页面的缩放系数=CSS 像素/设备独立像素，实际上说 页面的缩放系数=理想视口宽度/视觉视口宽度更为准确。

所以，当页面缩放比例为 100%时， CSS 像素=设备独立像素， 理想视口=视觉视口。

我们可以通过调用 screen.width/height 来获取理想视口大小。

4.4 Meta viewport

<meta> 元素表示那些不能由其它 HTML 元相关元素之一表示的任何元数据信息，它可以告诉浏览器如何解析页面。

我们可以借助 `<meta>`元素的 `viewport` 来帮助我们设置视口、缩放等，从而让移动端得到更好的展示效果。

```
<meta name="viewport" content="width=device-width; initial-scale=1; maximum-scale=1; minimum-scale=1; user-scalable=no;">
```

上面是 `viewport` 的一个配置，我们来看看它们的具体含义：

`Value`| 可能值| 描述 -|-|- `width`| 正整数或 `device-width` | 以 `pixels`（像素）为单位， 定义布局视口的宽度。 `height`| 正整数或 `device-height` | 以 `pixels`（像素）为单位， 定义布局视口的高度。 `initial-scale`| `0.0-10.0`|定义页面初始缩放比率。 `minimum-scale`| `0.0-10.0`|定义缩放的最小值；必须小于或等于 `maximum-scale` 的值。 `maximum-scale`| `0.0-10.0`|定义缩放的最大值；必须大于或等于 `minimum-scale` 的值。 `user-scalable`| 一个布尔值（ `yes` 或者 `no`）| 如果设置为 `no`，用户将不能放大或缩小网页。默认值为 yes。

4.5 移动端适配

为了在移动端让页面获得更好的显示效果，我们必须让布局视口、视觉视口都尽可能等于理想视口。

`device-width` 就等于理想视口的宽度，所以设置 `width=device-width` 就相当于让布局视口等于理想视口。

由于 `initial-scale=理想视口宽度/视觉视口宽度`，所以我们设置 `initial-scale=1;`就相当于让视觉视口等于理想视口。

这时，1 个 `CSS` 像素就等于 1 个设备独立像素，而且我们也是基于理想视口来进行布局的，所以呈现出来的页面布局在各种设备上都能大致相似。

4.6 缩放

上面提到 `width` 可以决定布局视口的宽度，实际上它并不是布局视口的唯一决定性因素，设置 `initial-scale` 也有肯能影响到布局视口，因为布局视口宽度取的是 `width` 和视觉视口宽度的最大值。

例如：若手机的理想视口宽度为 `400px`，设置 `width=device-width`，`initial-scale=2`，此时 `视觉视口宽度=理想视口宽度/initial-scale` 即 `200px`，布局视口取两者最大值即 `device-width` `400px`。

若设置 `width=device-width`， `initial-scale=0.5`，此时 `视觉视口宽度=理想视口宽度/initial-scale` 即 `800px`，布局视口取两者最大值即 `800px`。

4.7 获取浏览器大小

浏览器为我们提供的获取窗口大小的 `API` 有很多，下面我们再来对比一下：

- `window.innerHeight`：获取浏览器视觉视口高度（包括垂直滚动条）。
- `window.outerHeight`：获取浏览器窗口外部的高度。表示整个浏览器窗口的高度，包括侧边栏、窗口镶边和调正窗口大小的边框。
- `window.screen.Height`：获取获屏幕取理想视口高度，这个数值是固定的， `设备的分辨率/设备像素比`
- `window.screen.availHeight`：浏览器窗口可用的高度。
- `document.documentElement.clientHeight`：获取浏览器布局视口高度，包括内边距，但不包括垂直滚动条、边框和外边距。

- `document.documentElement.offsetHeight`：包括内边距、滚动条、边框和外边距。
- `document.documentElement.scrollHeight`：在不使用滚动条的情况下适合视口中的所有内容所需的最小宽度。测量方式与 `clientHeight` 相同：它包含元素的内边距，但不包括边框，外边距或垂直滚动条。